

BBBBBBBBBBBB		AAAAAAA		SSSSSSSSSS		RRRRRRRRRR		TTTTTTTTTTTT		LLL
BBBBBBBBBBBB		AAAAAAA		SSSSSSSSSS		RRRRRRRRRR		TTTTTTTTTTTT		LLL
BBBBBBBBBBBB		AAAAAAA		SSSSSSSSSS		RRRRRRRRRR		TTTTTTTTTTTT		LLL
BBB	BBB	AAA	AAA	SSS		RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA	SSS		RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA	SSS		RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA	SSS		RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA	SSS		RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA	SSS		RRR	RRR	TTT		LLL
BBBBBBBBBBBB		AAA	AAA	SSSSSSSS		RRRRRRRRRR		TTT		LLL
BBBBBBBBBBBB		AAA	AAA	SSSSSSSS		RRRRRRRRRR		TTT		LLL
BBBBBBBBBBBB		AAA	AAA	SSSSSSSS		RRRRRRRRRR		TTT		LLL
BBB	BBB	AAAAAAAAAAAA			SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAAAAAAAAAAA			SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAAAAAAAAAAA			SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA		SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA		SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA		SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA		SSS	RRR	RRR	TTT		LLL
BBB	BBB	AAA	AAA		SSS	RRR	RRR	TTT		LLL
BBBBBBBBBBBB		AAA	AAA	SSSSSSSSSS		RRR	RRR	TTT		LLLLLLLLLLLL
BBBBBBBBBBBB		AAA	AAA	SSSSSSSSSS		RRR	RRR	TTT		LLLLLLLLLLLL
BBBBBBBBBBBB		AAA	AAA	SSSSSSSSSS		RRR	RRR	TTT		LLLLLLLLLLLL

```

BBBBBBBBB      AAAAAA      SSSSSSSS      EEEEEEEEEEE      MM      MM      UU      UU      LL      PPPPPPPP
BBBBBBBBB      AAAAAA      SSSSSSSS      EEEEEEEEEEE      MM      MM      UU      UU      LL      PPPPPPPP
BB          BB  AA          AA  SS          SS          EE          EE          MMMM      MMMM      UU      UU      LL      PP          PP
BB          BB  AA          AA  SS          SS          EE          EE          MMMM      MMMM      UU      UU      LL      PP          PP
BB          BB  AA          AA  SS          SS          EE          EE          MM      MM      UU      UU      LL      PP          PP
BB          BB  AA          AA  SS          SS          EE          EE          MM      MM      UU      UU      LL      PP          PP
BBBBBBBBB      AA          AA      SSSSSS      EEEEEEEE      MM      MM      UU      UU      LL      PPPPPPPP
BBBBBBBBB      AA          AA      SSSSSS      EEEEEEEE      MM      MM      UU      UU      LL      PPPPPPPP
BB          BB  AAAAAAAAAA      SS          EE          EE          MM      MM      UU      UU      LL      PP
BB          BB  AAAAAAAAAA      SS          EE          EE          MM      MM      UU      UU      LL      PP
BB          BB  AA          AA      SS          EE          EE          MM      MM      UU      UU      LL      PP
BB          BB  AA          AA      SS          EE          EE          MM      MM      UU      UU      LL      PP
BBBBBBBBB      AA          AA      SSSSSSSS      EEEEEEEEEEE      MM      MM      UUUUUUUUUU      LLLLLLLLLL      PP
BBBBBBBBB      AA          AA      SSSSSSSS      EEEEEEEEEEE      MM      MM      UUUUUUUUUU      LLLLLLLLLL      PP

```

```

LL          IIIII
LL          IIIII
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LLLLLLLLLLL
LLLLLLLLLLL

          IIIII
          IIIII
          II
          II
          II
          II
          II
          II
          II
          II
          II
          II
          II
          II
          IIIII
          IIIII

          SSSSSSSS
          SSSSSSSS
          SS
          SS
          SS
          SS
          SSSSSS
          SSSSSS
          SS
          SS
          SS
          SS
          SSSSSSSS
          SSSSSSSS

```


(2) 52
(3) 94

DECLARATIONS
BAS\$EXTEND_MULP - Extended precision packed multiply

```
0000 1 .TITLE BAS$EXTEND_MULP - Extended precision packed decimal multiply
0000 2 .IDENT /1-003/ ; File: BASEMULP.MAR Edit: LB1003
0000 3
0000 4
0000 5 *****
0000 6 *
0000 7 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 * ALL RIGHTS RESERVED.
0000 10 *
0000 11 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 * TRANSFERRED.
0000 17 *
0000 18 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 * CORPORATION.
0000 21 *
0000 22 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 *
0000 25 *
0000 26 *****
0000 27
0000 28
0000 29 ++
0000 30 FACILITY: BASIC Language Support
0000 31
0000 32 ABSTRACT:
0000 33
0000 34 This module accepts two scaled packed decimal values and returns the
0000 35 correctly rounded (when required) result. This is done by computing
0000 36 a 62 digit result and then extracting the appropriate digits.
0000 37
0000 38 ENVIRONMENT: AST Reentrant
0000 39
0000 40 AUTHOR: Bob Hanek, CREATION DATE: 19-JAN-1981
0000 41
0000 42 MODIFIED BY:
0000 43
0000 44 1-001 - Original. RNH 01-JAN-1981
0000 45 1-002 - Change name to BAS$EXTEND_MULP and change errors to
0000 46 Decimal Error. PLL 12-Feb-1982
0000 47 1-003 - Add code to check value of flags field within BASIC frame
0000 48 for decimal overflow detection, and feed that info into
0000 49 the PSW. LB 14-May-1982
0000 50 --
```



```

0000 52      .SBTTL  DECLARATIONS
0000 53      :
0000 54      : LIBRARY MACRO CALLS:
0000 55      :
0000 56      :
0000 57      : EXTERNAL DECLARATIONS:
0000 58      :
0000 59      : .DSABL  GBL      : Force all external symbols to be declared
0000 60      : .EXTRN  BAS$K DECERR : Decimal overflow
0000 61      : .EXTRN  BAS$$STOP : Signal fatal errors
0000 62      : .EXTRN  BAS$HANDLER : BASIC condition handler
0000 63      :
0000 64      : MACROS:
0000 65      :
0000 66      : NONE
0000 67      :
0000 68      : EQUATED SYMBOLS:
0000 69      :
00000010 0000 70      LOW      = 16
00000020 0000 71      T1       = 32
00000030 0000 72      T2       = 48
00000040 0000 73      T3       = 64
00000050 0000 74      A1       = 80
0000005C 0000 75      A0       = 92
00000064 0000 76      B1       = 100
00000065 0000 77      B10      = 101
00000070 0000 78      B0       = 112
00000078 0000 79      DVFLAG   = 120
0000007C 0000 80      TOT       = 124
0000001C 0000 81      ONE      = ^X1C ; Packed decimal 1
0000 82      :
0000 83      :
0000 84      : OWN STORAGE:
0000 85      :
0000 86      : NONE
0000 87      :
0000 88      : PSECT DECLARATIONS:
0000 89      :
00000000 0000 90      .PSECT _BAS$CODE PIC, USR, CON, REL, LCL, SHR, -
0000 91      EXE, RD, NOWRT, LONG
0000 92

```



```
0000 94 .SBTTL BAS$EXTEND_MULP - Extended precision packed multiply
0000 95 :++
0000 96 : FUNCTIONAL DESCRIPTION:
0000 97 :
0000 98 : This module accepts two scaled packed decimal values and returns the
0000 99 : correctly rounded (when required) result. This is done by computing
0000 100 : a 62 digit result and then extracting the appropriate digits. In
0000 101 : particular, given two packed decimal values A' and B' of known length
0000 102 : we extend them to two 31 digit values A and B. Then let  $A = A1 \cdot 10^{15}$ 
0000 103 :  $+ A0$  and  $B = B1 \cdot 10^{15} + B0 = (B11 \cdot 10^{15} + B10) \cdot 10^{15} + B0$ , where  $0$ 
0000 104 :  $\leq A0, B0, B10 \leq 10^{15} - 1$ ,  $0 \leq A1 \leq 10^{16} - 1$  and  $0 \leq B11 \leq 9$ .
0000 105 : Then
0000 106 :
0000 107 :  $A \cdot B = (A1 \cdot 10^{15} + A0) \cdot (B1 \cdot 10^{15} + B0)$ 
0000 108 :  $= A1 \cdot B1 \cdot 10^{30} + (A1 \cdot B0 + A0 \cdot B1) \cdot 10^{15} + A0 \cdot B0$ 
0000 109 :  $= A1 \cdot (B11 \cdot 10^{15} + B10) \cdot 10^{30} + (A1 \cdot B0 + A0 \cdot B1) \cdot 10^{15} + A0 \cdot B0$ 
0000 110 :  $= A1 \cdot B11 \cdot 10^{45} + A1 \cdot B10 \cdot 10^{30} + (A1 \cdot B0 + A0 \cdot B1) \cdot 10^{15} + A0 \cdot B0$ .
0000 111 :
0000 112 : Note that all of the products of the form  $A_n \cdot B_m$  have absolute value
0000 113 : less than  $10^{31}$ , so that each product can be exactly computed by a MULP
0000 114 : instruction.
0000 115 :
0000 116 :
0000 117 :
0000 118 : CALLING SEQUENCE:
0000 119 :
0000 120 : CALL BAS$EXTEND_MULP(A.rp.dsd, B.rp.dsd, C.mp.dsd, RND_TRUNC.rb.v)
0000 121 :
0000 122 : FORMAL PARAMETERS:
0000 123 :
0000 124 : A and B are the scaled decimal source strings
0000 125 : C is the scaled decimal result string
0000 126 : RND_TRUNC is a flag indicating whether the final answer should be
0000 127 : rounded or truncated. RND_TRUNC = 5 gives rounded result and
0000 128 : RND_TRUNC = 0 gives truncated result.
0000 129 :
0000 130 : IMPLICIT INPUTS:
0000 131 :
0000 132 : NONE
0000 133 :
0000 134 : IMPLICIT OUTPUTS:
0000 135 :
0000 136 : NONE
0000 137 :
0000 138 : SIDE EFFECTS:
0000 139 :
0000 140 : Signals PACKED_OVERFLOW whenever the product is too large to fit in
0000 141 : the scaled decimal result operand
0000 142 :
0000 143 :--
```



```
03FC 0000 145 .ENTRY BAS$EXTEND_MULP, ^M<R2, R3, R4, R5, R6, R7, R8, R9>
      0002 146 ; Entry point
      0002 147 :
      0002 148 : Move parameters to registers R6 through R9
      0002 149 :
      56 04 AC 7D 0002 150 MOVQ 4(AP), R6 ; Move first two parameters to R6/R7
      58 0C AC 7D 0006 151 MOVQ 12(AP), R8 ; Move next two parameter to R8/R9
      000A 152 :
      000A 153 : REGISTER USAGE:
      000A 154 :
      000A 155 : R0 Through R5 are reserved for the packed decimal instruction
      000A 156 : R6: Pointer to A descriptor
      000A 157 : R7: Pointer to B descriptor
      000A 158 : R8: Pointer to C descriptor
      000A 159 : R9: Round/trun flag
      000A 160 :
      000A 161 :
      000A 162 :
      000A 163 : Allocate stack storage and turn off decimal overflow reporting.
      000A 164 : Check the setting of the decimal overflow bit in the flags field
      000A 165 : within the BASIC frame. Use that value to set the PSL decimal
      000A 166 : overflow trap setting (enable or disable).
      000A 167 :
      5E 0000007C 8F C2 000A 168 SUBL #TOT, SP ; Allocate 26 longwords on stack
      54 0C AD D0 0011 169 MOVL 12(FP), R4 ; Fetch the saved frame pointer
      55 00000000 GF DE 0015 170 MOVAL G^BAS$HANDLER, R5 ; Fetch addr of BASIC handler
      55 64 D1 001C 171 CMPL 0(R4), R5 ; Check if this is a BASIC frame
      0080 8F B8 0021 172 BEQL 5$ ; Branch if a BASIC frame
      17 11 0025 173 BISPSW #^X80 ; Else, set DV bit in PSL
      54 E6 A4 B0 0027 174 BRB 15$
      54 FBFF 8F AA 002B 175 5$: MOVW -26(R4), R4 ; Fetch the flags word
      54 54 B5 0030 176 BICW #^XFBFF, R4 ; Mask out all but DV bit
      0080 8F B8 0034 177 TSTW R4 ; Check if DV was set in BASIC frame
      04 11 0038 178 BEQL 10$ ; Br if not enabled
      0080 8F B8 0034 179 BISPSW #^X80 ; Set DV bit in PSL (logical OR)
      0080 8F B9 003A 180 BRB 15$ ; Continue as usual
      78 AE FF7F 8F AA 003E 181 10$: BICPSW #^X80 ; Clear DV bit in PSL (logical AND)
      0080 8F B9 0047 182 15$: MOVPSL DVFLAG(SP) ; Save current PSL
      78 AE FF7F 8F AA 0041 183 BICW #^XFF7F, DVFLAG(SP) ; Save only decimal overflow bit
      0080 8F B9 0047 184 BICPSW #^X80 ; Turn off decimal overflow reporting
      004B 185 :
      004B 186 : Break B into its high and low parts
      004B 187 :
      6E 1F 00 04 B7 67 00 F8 004B 188 ASHP #0, (R7), @4(R7), #0, #31, (SP) ; Extend B to 31 digits
      70 AE 08 AE 7D 0053 189 MOVQ 8(SP), B0(SP) ; Set B0 to low 15 digits of B
      10 00 6E 1F F1 8F F8 0058 190 ASHP #-15, #31, (SP), #0, #16, B1(SP); Set B1 to high 16 digits of B
      64 AE 005F 191 :
      0061 192 : Break A into its high and low parts
      0061 193 :
      6E 1F 00 04 B6 66 00 F8 0061 194 ASHP #0, (R6), @4(R6), #0, #31, (SP) ; Extend A to 31 digits
      5C AE 08 AE 7D 0069 195 MOVQ 8(SP), A0(SP) ; Set A0 to low 15 digits of A
      10 00 6E 1F F1 8F F8 006E 196 ASHP #-15, #31, (SP), #0, #16, A1(SP); Set A1 to high 16 digits of A
      50 AE 0075 197 :
      0077 198 : Compute A0*B0 and store in the low digits of the result.
      0077 199 :
      0077 199 :
```



```
1F 70 AE 0F 5C AE 10 AE 25 0077 200 MULP #15, A0(SP), #15, B0(SP), #31, LOW(SP)
                                007F
                                0081 201 :
                                0081 202 : Compute A0*B1 and store the results in a 31 digit temporary
                                0081 203 :
1F 64 AE 10 5C AE 20 AE 25 0081 204 MULP #15, A0(SP), #16, B1(SP), #31, T1(SP)
                                0089
                                008B 205 :
                                008B 206 : Add low 16 digits of A0*B1 to low digits of the result and move the high
                                008B 207 : 15 digits of A0*B1 to high digits of the result
                                008B 208 :
                                008B 209 :
                                0092
                                0094 210 ASHP #15, #31, T1(SP), #0, #31, T2(SP) ; T2 = Lo 16 digits of A0*B1
                                009C
                                009D 211 ASHP #-16, #31, T1(SP), #0, #31, (SP) ; (SP) = Hi 15 digits of A0*B1
                                00A4 212
                                00A6 213 ADDP #31, T2(SP), #31, LOW(SP) ; Add lo 16 digits to lo
                                00AC 214 BVC 1$ ; digits of the result and
                                00AC 215 ADDP #1, #ONE, #31, (SP) ; and propagate the carry
                                00AC 216
                                00AC 217 1$: MULP #16, A1(SP), #15, B0(SP), #31, T1(SP)
                                00B4
                                00B6 218 :
                                00B6 219 : Add low 16 digits of A1*B0 to low digits of the result and add the high
                                00B6 220 : 15 digits of A1*B0 to high digits of the result
                                00B6 221 :
                                00B6 222 :
                                00BD 223 ASHP #15, #31, T1(SP), #0, #31, T2(SP) ; T2 = lo 16 digits of A1*B0
                                00C7 224 ASHP #-16, #31, T1(SP), #0, #31, T3(SP); T3 = hi 15 digits of A1*B0
                                00C9 225 ADDP #31, T2(SP), #31, LOW(SP) ; Add lo digits to lo digits
                                00D0 226 BVC 2$ ; Branch if no carry
                                00D2 227 ADDP #1, #ONE, #31, T3(SP) ; Propagate carry
                                00D9 228 2$: ADDP #31, T3(SP), #31, (SP) ; Add hi digits to hi digits
                                00DF 229 :
                                00DF 230 : Compute B11*10^15 and A1*B10
                                00DF 231 :
                                00E5 232 MOVP #15, B10(SP), T3(SP) ; T3 = B10
                                00EC 233 SUBP #15, T3(SP), #16, B1(SP) ; B1 = B11*10^15
                                00F4 234 MULP #15, T3(SP), #16, A1(SP), #31, T1(SP) ; T1 = A1*B10
                                00F6 235 :
                                00F6 236 : Add low digit of A1*B10 to low digits of result and add high 30 digits of
                                00F6 237 : A1*B10 to high digits of result.
                                00F6 238 :
                                00FD 239 ASHP #30, #31, T1(SP), #0, #31, T2(SP) ; T2 = lo digit of A1*B10
                                0107 240 ASHP #-1, #31, T1(SP), #0, #31, T3(SP) ; T3 = hi 30 digits of A1*B10
                                0109 241 ADDP #31, T2(SP), #31, LOW(SP) ; Add lo digit to lo digits
                                0110 242 BVC 3$ ; Branch if no carry
                                0112 243 ADDP #1, #ONE, #31, (SP) ; Propagate carry
                                0118 244 3$: ADDP #31, T3(SP), #31, (SP) ; Add hi digits to hi digits
                                011E 245 :
                                011E 246 : Compute A1*B11*10^14 and add to high digits of result
```


BAS\$EXTEND_MULP
1-003

D 4
- Extended precision packed decimal multi 15-SEP-1984 23:38:35 VAX/VMS Macro V04-00
BAS\$EXTEND_MULP - Extended precision pac 6-SEP-1984 10:24:29 [BASRTL.SRC]BASEMULP.MAR;1

Page 6
(4)

```

OF 00 64 AE 10 FF 8F F8 011E 247 ASHP # -1, #16, B1(SP), #0, #15, T3(SP) ; T3 = B11*10^14
1F 50 AE 10 40 AE 25 0126 248 MULP #15, T3(SP), #16, A1(SP), #31, T1(SP) ; T1 = A1*B11*10^14
6E 1F 20 AE 1F 20 0130 249 ADDP #31, T1(SP), #31, (SP) ; Add to high digits
0138 250 :
0138 251 : Determine the number of place we need to shift the result.
0138 252 :
55 08 A7 08 A6 81 0138 253 ADDB3 8(R6), 8(R7), R5 ; R5 = total # of fraction digits
55 08 A8 82 013E 254 SUBB 8(R8), R5 ; R5 = # of extra fractional digits
0142 255 :
0142 256 : Shift the result the necessary number of digits, round if necessary, and
0142 257 : Store the result in the destination operand.
0142 258 :
1F 59 10 AE 1F 55 F8 0142 259 ASHP R5, #31, LOW(SP), R9, #31, T1(SP) ; T1 = final result lo digits
20 AE 55 1F 80 0149 260 ADDB #31, R5 ; R5 = shift for high digits
30 AE 1F 00 6E 1F 55 F8 014E 261 ASHP R5, #31, (SP), #0, #31, T2(SP) ; T2 = final result hi digits
30 AE 1F 20 AE 1F 20 0156 262 ADDP #31, T1(SP), #31, T2(SP) ; T2 = Final result
16 1D 015D 263 BVS OVERFLOW ; Branch if overflow
68 00 30 AE 1F 00 F8 015F 264 ASHP #0, #31, T2(SP), #0, (R8), @4(R8) ; Store result in destination
04 B8 0166 265 BVS OVERFLOW ; Branch if overflow
08 1D 0168 266 RETURN: BISPSW DVFLAG(SP) ; Restore decimal overflow flag
78 AE B8 016A 267 ADDL #TOT, SP ; Deallocate stack storage
5E 0000007C 8F C0 016D 268 RET ; Return
04 0174 269 :
0175 270 :
0175 271 :
0175 272 OVERFLOW:
78 AE B5 0175 273 TSTW DVFLAG(SP) ; Did caller enable Dec ovfl reporting?
F0 13 0178 274 BEQL RETURN ; Branch if reporting not enabled
78 AE B8 017A 275 BISPSW DVFLAG(SP) ; Restore decimal overflow flag
5E 0000007C 8F C0 017D 276 ADDL #TOT, SP ; Deallocate stack storage
00000000 8F DD 0184 277 PUSHL #BAS$K_DECERR ; Signal overflow
00000000 GF 01 FB 018A 278 CALLS #1, G^BAS$$$STOP
04 0191 279 RET
0192 280
0192 281
0192 282 .END ; End of module BAS$EXTEND_MULP

```


BAS\$EXTEND MULP
Symbol table

A0	=	0000005C		
A1	=	00000050		
B0	=	00000070		
B1	=	00000064		
B10	=	00000065		
BAS\$\$STOP	*****		X	00
BAS\$EXTEND MULP	00000000		RG	01
BAS\$HANDLER	*****		X	00
BAS\$K DECERR	*****		X	00
DVFLAG	=	00000078		
LOW	=	00000010		
ONE	=	0000001C		
OVERFLOW	00000175		R	01
RETURN	0000016A		R	01
T1	=	00000020		
T2	=	00000030		
T3	=	00000040		
TOT	=	0000007C		

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes													
ABS	00000000 (0.)	00 (0.)	NOPIC	USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE			
_BAS\$CODE	00000192 (402.)	01 (1.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	LONG			

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	30	00:00:00.07	00:00:00.30
Command processing	120	00:00:00.50	00:00:01.83
Pass 1	75	00:00:00.88	00:00:02.58
Symbol table sort	0	00:00:00.01	00:00:00.01
Pass 2	67	00:00:00.65	00:00:01.17
Symbol table output	3	00:00:00.02	00:00:00.08
Psect synopsis output	2	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	299	00:00:02.16	00:00:05.99

The working set limit was 750 pages.
5342 bytes (11 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 18 non-local and 6 local symbols.
282 source lines were read in Pass 1, producing 11 object records in Pass 2.
0 pages of virtual memory were used to define 0 macros.

! Macro library statistics !

Macro library name

Macros defined

_S255\$DUA28:[SYSLIB]STARLET.MLB;2

0

0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:BASEMULP/OBJ=OBJ\$:BASEMULP MSRC\$:BASEMULP/UPDATE=(ENH\$:BASEMULP)

0022 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

BASEDDFS
LIS

BASERROR
LIS

BASEDDDF
LIS

BASEDIT
LIS

BASEND
LIS

BASEDUP
LIS

BASEMULP
LIS

BASEDDGB
LIS

BASERTXT
LIS